SBIR - 04.02 - 8095
Release Date: 8-6-92

IN-61-CR
133342
P - 21

ADEPT

AUTOMATED DESIGN EXPERT PROGRAMMER'S MANUAL

Prepared by:
John P. Frazier
U.V.L. Narayana
- of -
Autodesk, Inc.
1303 Hightower Trail, Suite 170
Atlanta, GA 30350

Prepared for :
NASA - Lewis Research Center
Cleveland, OH 44135

Contract # NAS3-25150
An Expert System for Finite Element Modeling

June 8, 1990

ADEPT PROGRAMMER'S MANUAL

TABLE OF CONTENTS

PROGRAM LAYOUT

Adept is an application program that guides a design engineer in the creation of an appropriate finite element mesh from the starting point of having a solid. **Adept** is dependent upon AutoSolid for the solid geometry and windowing capabilities. The transfer of information between AutoSolid and **Adept** is accomplished through the AutoSolid Programming Interface (API) included within AutoSolid. Integrated into **Adept** is an expert system possessing knowledge about the assumptions and the methodology associated with finite element modeling and analysis. The expert system shell being used is CLIPS from the NASA Johnson Space Flight Center. The knowledge is in the form of a rule-base that drives the accumulation of needed information from both the user and AutoSolid. Decision branches are made based on the facts in the working memory defining the specific situation.

Conceptually, **Adept** consists of three basic parts. In the directory "nasa" are the following directories - clips, head, and envdes, the environment descriptor. Clips is the expert system portion of the program and the environment descriptor is a group of routines that creates menus and contains algorithms that determine characteristics of the solid being studied. The "head" directory describes the data structures used throughout the "envdes" directory. These structures are the method by which separate pieces of information are grouped. In the "envdes" directory are three additional directories. The "geom" directory deals with issues of categorizing and altering the geometry of the original solid. The "feat" directory has files which are responsible for defining both simple and complex features and for applying loads and boundary conditions to these features. The "misc" directory is the miscellaneous area where additional functions reside. Included in this area are routines to determine the purpose and resources for the analysis and routines essential to many of the meshing abilities of **Adept.**

PROGRAM EXAMPLES

Adept includes a run-time version of CLIPS and an associated rule-base. During the execution of an **Adept** session facts are asserted defining the current status. These facts might describe characteristics of the solid being considered for finite element modeling or the facts might concern what menu is being shown to the user. The presence of facts determine what rules fire. The firing of these rules can have the following consequences. Existing facts can be retracted where facts are removed from the working memory. New facts could be generated. Functions can be called to obtain additional information about the modeling situation.

All the rules reside in a file named rules.clp. This file is located in the clips directory. Excerpts from that file will be inspected in detail so the user will understand how to alter the existing rule base if desired. Information will also be supplied in order that the user can recompile an altered version of **Adept** to visualize the consequences of changing or adding rules.

A short introduction to clips will be given here. The user is advised to study the CLIPS USER GUIDE and CLIPS REFERENCE MANUAL from the Johnson Space Center for in-depth instruction on programming in CLIPS. Let's begin with a simple clips rule that will generate a new fact from some existing facts. This should give the user an understanding of how the working memory changes. In the clips directory is the file "rules.clp". In this file are the rules, including:

```
(defrule Mirror:x ""
   (declare (salience 20))
   (Simplification Checks)
   (Geom Mirx True)
   (Lbc Mirx True)
=>
   (assert (Reducex Yes))
)
```

For this rule to fire, the facts "Simplifications Checks", "Geom Mirx True", and "LBC Mirx True" must exist. The declare command is a method of having this particular rule fire before or after a rule of otherwise equal importance. A salience number can be set from -10,000 to +10,000. Larger numbers place rules of otherwise equal level in position to be fired first. When this rule named "Mirror:x" fires it creates a new fact. The working memory now includes the fact "Reducex Yes".

With this rule we have determined that in the case of the geometry having a symmetry in the x-direction and the loads and boundary conditions having the same symmetry in the x-direction we will keep a fact in the memory which states we wish to reduce the solid by cutting the solid in half at the plane X=0. We have in this case only stated a fact, we have not called any function to perform the reduction of the solid.

In the next example is a case where the presence of facts fire a rule and then the facts are retracted. This enables the user to keep the working memory free of extraneous information. The retraction of facts also allows the user to restate the same facts and refire the rule in order to use a section of the program more than one time. Here is the rule "Get:NewForces":

```
(defrule Get:NewForces ""
    (Stage 0)
    ?f1<-(Loadbcs Obtained)
    ?f2<-(Loadset Bad)
=>
    (retract ?f1 ?f2)
    (assert (Redo Loadbcs))
)
```

If the facts "Stage 0", "Loadbcs Obtained", and "Loadset Bad" exist, the rule will fire. The retract statement omits the facts "Loadbcs Obtained" and "Loadset Bad". Following this operation another fact is asserted. If the loads and boundary conditions are obtained and if checks have been done to determine that the loadset is not acceptable then the program proceeds to the section to get a new or updated set of loads and boundary conditions.

The following example includes a call to a user generated function. At the end of the rule is the routine "clips-over". Look in the file "clips/usrfuncs.c" to find the proper way to define all user supplied functions. Look in the file "envdes/misc/mk_mesh.c" to find the actual "c" function "clips_over()". All of the user defined functions in "rules.clp" are generated in roughly the same way.

```
(defrule Clips:Finished ""
    ?f1<-(Mesh Exists)
=>
    (retract ?f1)
    (clips-over)
)
```

A more advanced example shows the use of a supplied function that returns a specific value to CLIPS.

```
(defrule Nostress:Option ""
   (Stage 0)
   (Loadbcs Obtained)
   ?f2<-(Give User Warning 1)
=>
   (retract ?f2)
   (bind ?retval (lbc-accept-option))
   (if (= ?retval 0)
      then
         (assert (Loadset Bad))
      else
         (assert (Stress Generated))
      )
)
```

In this case the function "lbc-accept-option" captures the return value and stores the integer returned in the variable "retval". Comparisons are then performed to see which fact should be asserted. The actual function "lbc_accept_opt()" is contained in "envdes/geom/sset.c". Also in this rule is the method to perform comparisons. The line with "if (= ?retval 0)" shows the format for comparing a variable to some constant value. The constant could have just as easily been another constant. Variables are only good inside a rule. If a variable value were needed in another rule it could be retained through the use of an asserted fact. The following rule gives an example of this.

```
(defrule Reduce:Solid ""
   ?f1<-(Reduce Solid)
   ?f2<-(Reducex ?x)
   ?f3<-(Reducey ?y)
   ?f4<-(Reducez ?z)
=>
   (reduce-solid ?x ?y ?z)
   (symm-bc ?x ?y ?z)
   (retract ?f1)
   (assert (Set Elemtype))
)
```

In this rule the variables "x", "y", and "z" are variables that are defined to be either Yes or No. This occurs because if the fact "Reducex -" exists, the second word is always "Yes" or "No". The same is true for the other directions. In this way the variable "x" is set to "Yes" or "No". Also shown in this rule is the method to pass arguments to user supplied routines. In the function "reduce-solid" three arguments are passed. Refer to "envdes/misc/s_mparm2.c" to find the function "reduce_solid()" and an example of how to pass arguments to functions.

COMPILING A NEW VERSION OF ADEPT

In the event the user wishes to add facts or routines to **Adept** this section will diagram the method by which a new version of **Adept** can be generated. **Adept** is an application program that attaches to AutoSolid. All information which passes between AutoSolid and **Adept** is accomplished through the AutoSolid Programming Interface (API). When a user supplied program is to be attached to AutoSolid, the user gives a command similar to:

setenv ASUSRPROC1 /files/home/users/cadre/nasa/envdes/adept

In this case the directory location of the adept executable is specified. If the user wished to detach the program from AutoSolid this can be accomplished by the command:

setenv ASUSRPROC1 noproc

More on this subject can be found in the Introduction to the AutoSolid Programming Interface Manual.

If any files which reside in "nasa/envdes/{feat, geom, or misc}" are changed, the user will need to issue the operating system command 'make' while in the directory where a change has occurred. If any changes have been made to either "nasa/clips/rules.clp" or to "nasa/clips/usrfuncs.c" then the user will have to update the compiled versions in the directory clips. This can be accomplished with the following commands. Enter the clips directory. Then type:

make<cr> <cr> is carriage return. This matches the functions called for in rules.clp and usrfuncs.c with those in feat, geom, and misc. All requested functions need to exist somewhere in the code.

CLIPS<cr> This command starts the stand alone version of clips.
(load "rules.clp")<cr> This loads the rules.clp file.
(rules-to-c "rules.c" 1)<cr> This cause clips to generate c code directly from the rules.
(exit)<cr> This exits the stand alone version of clips.

make lib<cr> This generates the object files in the clips directory.

Now return to the envdes directory. The 'make' command in this directory will now generate the executable **Adept**. This completes the operation for creating an updated version of the **Adept** program.

CODING STRUCTURE FOR ROUTINES

This section is included in an effort to give the user an overview of the structure of the **Adept** program. A brief description of the file structure is given in case the user would like to make programming changes to the current **Adept** system. As stated previously, **Adept** consists of three portions, a data structure directory (head), an environment descriptor directory (envdes), and an expert system directory (clips).

Table 1 is a summary of the most important structures defined in the "nasa/head" directory.

Included in the "nasa/envdes" directory are three separate directories. These are the "feat", "geom", and "misc" directories. Tables 2 through 4 define the file contents of these directories with a short summary of the processes that are handled within each file. This portion of the program deals exclusively with the description of the solid and the attached attributes. In these routines are many api (AutoSolid Programming Interface) calls back to the geometric database for the purpose of obtaining specific facts about the solid. Refer to the AutoSolid Programming Interface Reference Manual for an in-depth description of building and attaching an application program for AutoSolid.

The expert system portion of the program is contained within the "nasa/clips" directory. There are many files in this directory with the vast majority of these being specific to the generic version of CLIPS. The only files the user needs to change in the process of generating an altered version of **Adept** are "usrfuncs.c" and "rules.clp". The file "usrfuncs.c" handles the association of clips defined routines with the actual "c" routines located in the environment descriptor portion of the software. The "rules.clp" file contains the clips rules which direct the procedure of the building the finite element mesh from the original solid. Very few arithmetic calculations are performed in the rules file. The environment descriptor is heavily used to obtain facts about the solid and its loading and boundary conditions. The rules control the information gathering and the determinations that are made once the characteristics of the specific situation have been classified. Refer to either the USER'S MANUAL or the THEORETICAL MANUAL for an overview of the procedure used to create the finite element mesh from the original solid.

TABLE 1. CONTENTS OF DIRECTORY HEAD

file name = na_bfil.h

```
/* Solid data structure */

typedef struct na_sol
{
        struct na_face *fac;        /* linked list of faces */
        struct na_edge *edg;        /* linked list of edges */
        struct na_vert *ver;        /* linked list of verts */
        struct na_sol  *nxt;        /* next solid */
        ap_Mprop *mprop;            /* mass props pointer */
        struct na_feat *feat;       /* pointer to a list of assoc. feats */
        float s_area;               /* total surface area of the solid */
        ap_Solid sol;               /* original solid id */
        ap_Solid sol2;              /* simplified solid id */
        ap_Sid str;                 /* graphics str */
        na_Extrevp extrev;          /* sweep info. pointer */
        na_Mirplnp msymm;           /* geometric symm. info */
        na_Repsymp rsymm;           /* repetitive symm */
        short lbmsymm;              /* code for lbc symm */
        na_Ext_rev lbswp;           /* code for lbc sweep */
        short lbrsymm;              /* code for lbc repsym */
        short dimn;                 /* FE model dimension code */
        short curv;                 /* FE model curvature code */
        short lbc_dir;              /* code for lbc dirn. */
        short purp;                 /* code for purpose of anal. */
        short hware;                /* code for hardware type */
        short swart;                /* code for software type */
        short swarl;                /* code for software lim. */
        short time;                 /* code for time of analysis execution */
        short msh_density;          /* code for mesh density */
        ap_ElemListp elmlst;        /* linked list of assoc. elements */
} na_Sol, *na_Solp;


/* Face data structure */

typedef struct na_face
{
        ap_Fac l_face_id;               /* AutoSolid face id */
        ap_FaceInfo l_face;             /* Information about face */
        struct na_edgi *edgi;           /* linked list of edges */
        struct na_veri *veri;           /* linked list of verts */
        struct na_face *l_facenext;     /* next face */
        short util;                     /* utility code */
        struct na_facl *sfac;           /* sub-face list */
        Bool is_mesh;                   /* flag if surface to be meshed for 2d */
        Bool shell_tested;              /* flag to check if XXXXX */
        ap_Sid str;                     /* graphics str */
        ap_ElemListp elmlst;            /* element list to be assoc. with face */
        struct ap_pmesh *pmesh;         /* pointer to pmesh of the surf. */
        ap_Real box[6];                 /* face box */
} na_Fac, *na_Facp;


/* Linked list for sub-faces */

typedef struct na_facl
{
        struct na_faci *faci;           /* pointer to the sub-face */
        struct na_facl *nxt;            /* pointer to the next sub-face */
} na_Facl, *na_Faclp;


/* Sub-face data structure */

typedef struct na_faci
{
        na_Facp fac;                    /* back pointer to the original face */
        struct na_faci *nxt;            /* next fac */
        short util;                     /* utility code for display */
        float s_perim;                  /* perimeter of the sub-face */
        float s_area;                   /* area of the sub-face */
        ap_Sid str;                     /* graphics structure */
        Lbc_list lbc;                   /* lbcs for the sub-face */
        short feat_imp;                 /* feature importance code */
        short msh_density;              /* mesh_density level */
        struct na_feat *feat;           /* linked list of features it belongs to */
        short is_cavfac;                /* flag to det. if it is a concave face */
} na_Faci, *na_Facip;


/* Edge data structure */

typedef struct na_edge
{
        ap_Edg l_edge_id;               /* AutoSolid edge id */
        ap_EdgeInfo l_edge;             /* edge information */
        struct na_face *fac[2];         /* the two bounding faces */
        struct na_vert *ver[2];         /* the two bounding edges */
        struct na_edge *l_edgenext;     /* next edge */
        short util;                     /* utility code */
        ap_Sid str;                     /* graphics structure */
        Lbc_list lbc;                   /* lbc list for the edge */
        short feat_imp;                 /* feature importance code */
        short msh_density;              /* mesh density level */
        ap_ElemListp elmlst;            /* list of assoc. elements */
        ap_Real box[6];                 /* edge box */
} na_Edg, *na_Edgp;


/* Linked list for edges */
```

```
typedef struct na_edgi
{
        na_Edgp edg;                   /* pointer to the underlying edge */
        struct na_edgi *nxt;           /* pointer to the next data structure */
        short util;                    /* utility code */
} na_Edgi, *na_Edgip;


/* Vertex data structure */

typedef struct na_vert
{
        ap_Ver l_vert_id;              /* AutoSolid vertex id */
        ap_VerPt l_vert;               /* vertex coord info. */
        struct na_faci *faci;          /* linked list of assoc. faces */
        struct na_edgi *edgi;          /* linked list of assoc. edges */
        struct na_vert *l_vertnext;    /* next vertex */
        short util;                    /* utility code */
        short numedg;                  /* number of assoc. edges */
        short numcav;                  /* number of assoc. concave edges */
        ap_Sid str;                    /* graphics structure */
        Lbc_list lbc;                  /* lbc list for the vertex */
        short feat_imp;                /* feature importance code */
        short msh_density;             /* mesh density level */
        ap_ElemListp elmlst;           /* list of assoc. elements */
} na_Ver, *na_Verp;


/* Linked list for vertices */

typedef struct na_veri
{
        na_Verp ver;                   /* underlying vertex */
        struct na_veri *nxt;           /* pointer to the next data structure */
} na_Veri, *na_Verip;


typedef ap_VertList *ap_VertListp;     /* Pointer to the API vertex list */
typedef ap_EdgeList *ap_EdgeListp;     /* Pointer to the API edge list */
typedef ap_FaceList *ap_FaceListp;     /* Pointer to the API face list */
```

file name = na_feat.h

```
typedef enum
{
        STEP =0,
        NOTCH,
        BLIND_NOTCH,
        BLIND_STEP,
        SLOT,
        BLIND_SLOT,
        POCKET,
        HOLE,
        PROTRUSION,
        BRIDGE,
        CAVITY,
        STOCK,
        DISJOINT,
        NON_MANIFOLD,
        UNKNOWN,
        USED,
        AMBIG
} Feat_type;


/* Complex feature data structure */

typedef struct na_feat
{
        Feat_type ftype;          /* feature type */
        na_Facip facp;            /* list of assoc. sub-faces */
        na_Edgip edgp;            /* list of assoc. edges */
        struct na_feat *nxt;      /* next edge */
        na_Facp cavfac;           /* pointer to underlying concave face */
        int numcavedg;            /* number of concave edges */
        int numcavfac;            /* number of concave faces */
        na_Solp fsol;             /* the assoc. prim with na_feat */
        float s_area;             /* surface area of the feature */
        ap_Sid str;               /* graphics struct of the feature */
        short feat_imp;           /* feature importance code */
        short msh_density;        /* mesh density code */
} na_Feat, *na_Featp;
```

file name = na_symm.h

```
typedef enum
{
        NONE =0,
        EXTN,
        REVN,
        BOTH,
        UNKN
} na_Ext_rev;


/* Extrusion/Revolution info data structure */

typedef struct extrev
{
        na_Ext_rev code;          /*     extrusion/revolution code */
        ap_Vector3d eaxis;        /* extrusion axis orientation */
        ap_Wcpt ecen;             /* extrusion axis location */
        ap_Vector3d raxis;        /* revolution axis orientation */
        ap_Wcpt rcen;             /* revolution axis location */
        Bool sphere;              /* code for sphere; multiple raxis */
        Bool cuboid;              /* code for cube; multiple eaxis */
        na_Ext_rev test;          /* code to check if already tested */
} na_Extrev, *na_Extrevp;


/* featids */

typedef enum
{
        NORS =0,
        LINE,
        CIRC,
        BORS,
        UNRS
} na_Rep_sym;


/* Repetitive symmetry data structure */

typedef struct repsym
{
        na_Rep_sym code;          /* repetitive symmetry code */
        ap_Vector3d laxis;        /* linear symm. axis orientation */
        ap_Wcpt lcen;             /* linear symm. axis location */
        ap_Vector3d raxis;        /* circular symm. axis orientation */
        ap_Wcpt rcen;             /* circular symm. axis location */
        na_Rep_sym test;          /* code to check if already tested */
} na_Repsym, *na_Repsymp;


/* Mirror plane data structure */
```

```
typedef struct mirpln
{
        short code;                 /* mirror plane code for X/Y/Z symm */
        ap_Vector3d p1axis;         /* redundant field for 1st mplane */
        ap_Vector3d p2axis;         /* redundant field for 2nd mplane */
        ap_Vector3d p3axis;         /* redundant field for 3rd mplane */
        ap_Wcpt pcen;               /* redundant field for centroid of mplanes */
        short test;                 /* code to check if already tested */
} na_Mirpln, *na_Mirplnp;
```

file name = na_lbc.h

```
#define EPSILON .000001

typedef enum
{
        BC_FIXED,
        BC_FREE,
        BC_LIMITED
} Bc_free ;


/* Boundary condition data structure */

typedef struct bc
{
        Bc_free bc_free ;           /* boundary condition type */
        Real bc_lim[3] ;            /* limited displacement vector */
} Bc ;


typedef enum
{
        LD_FORCE,
        LD_PRESS,
        LD_TEMP,
        LD_RACCEL,
        LD_LACCEL,
        LD_RVEL,
        LD_RTEMP,
        LD_UTEMP
} Ld_type ;


/* load data structure */

typedef struct load
{
        Ld_type ld_type ;           /* load type */
        ap_Real ld_mag ;            /* load magnitude */
        ap_Vector3d ld_vec ;        /* load vector */
} Load ;


/* lbc data structure */

typedef struct lbc_list
{
        Bool lbc_isload ;           /* load or bc */
        int lbc_setndx ;            /* index into load/bc group */
        union
        {
                Load lbc_load ;     /* load data */
                Bc lbc_bc[6] ;                  /* bc data */
        } lbc_un ;
        Featid lbc_fid ;                        /* feature with which load/bc associated */
        struct lbc_list *lbc_next ;             /* next lbc_list struct */
        ap_Sid str;                             /* graphics structure */
} *Lbc_list ;


typedef enum
{
        EL_LTRIAN,
        EL_QTRIAN,
        EL_LSHELL,
        EL_QSHELL,
        EL_LTETRA,
        EL_QTETRA
} Elem_type ;                                   /* element type */


typedef ap_ElemList *ap_ElemListp;              /* linked list of elements */
typedef ap_FeatList *ap_FeatListp;              /* linked list of features */
```

file name = na_node.h

```
/* Tree node data structure */

typedef struct na_node
{
        long na_type;                       /* node type */
        long na_boolop;                     /* node */
        struct na_node *na_left;            /* left child */
        struct na_node *na_right;           /* right child */
        struct na_node *na_parent;          /* back pointer to the parent */
        long na_solid;                      /* solid id */
        float na_real;                      /* optional float field */
        ap_Trans3d mat;                     /* rigid motion of the node */
} na_Nnod, *na_Nnodp;


/* type codes for na_type */

#define NASPPRIM 1000L
#define NATYPE 100L
#define NATBLO 103L
#define NATCON 402L
#define NATCYL 202L
#define NATSPH 301L
#define NATTOR 502L
#define NATWED 603L
#define NATMET 900L
#define NAHALF 8000L
#define NAMPLA 8001L
#define NAMCON 8002L
#define NAMCYL 8003L
#define NAMSPH 8004L
#define NAMTOR 8005L
#define NAMBOX 8006L
#define NANVAL 2000L
#define NANRAD 2001L
#define NANHGT 2002L
#define NANXVL 2003L
#define NANYVL 2004L
#define NANZVL 2005L
#define NANMAJ 2006L
#define NANMIN 2007L
#define NANALP 2008L
#define NANCMP 2009L
#define NAOPER 4000L
#define NAOUNI 4001L
#define NAODIF 4002L
#define NAOINT 4003L
#define NAOASB 4004L
#define NAOCOM 4005L
#define NAANULL 0L
```

TABLE 2. CONTENTS OF DIRECTORY FEAT

na_bbfil.c	* Create the boundary file data structs for the solid *
X na_bfeat.c	* Create user defined complex features *
na_eqedg.c	* Geometry checks to see if two edges are the same edge *
na_eqhlf.c	* Geometry checks to see if two half-spaces are the same *
na_feat.c	* Controls definition of complex features from original solid * * X Automatic, X User-defined, Primitive classification *
X na_feat1.c	* Automatic complex feature category determination *
X na_feat2.c	* Flag certain simple features for display purposes *
X na_feat3.c	* Create meta-primitives for features *
X na_feat4.c	* Highlight the previously obtained complex features *
na_lbc.c	* Adds/deletes lbcs to features - also shows simple features *
na_pfeat.c	* Primitives are classified as complex features *
na_pft1.c	* Create an updated tree for solid in terms of features *
na_pft2.c	* Highlight complex features and match new tree with the old *
X na_prot.c	* Track faces attached to concave/convex edges of features *
na_reori.c	* Relocate the solid's centroid and match principal axes to global axes *
na_supt.c	* Create polymesh data structs for lbc visualization *

The features and routines marked with X are de-linked from ADEPT for now due to efficiency and stability reasons.

TABLE 3. CONTENTS OF DIRECTORY GEOM

apprm.c	* determine new rigid motion for a new orientation of the solid *
apprm2.c	* apply the rigid motion to lbcs *
apprm3.c	* update box size, features etc. with new rigid motion *
cgeom.c	* menu control for geometric characterization of solid *
cgeom2.c	* menu control for extrusion/revolution and mirror symmetries *
cgeom3.c	* filtering complex features from original solid *
clbc.c	* classification of characteristics of loads and boundary conds *
clbc2.c	* algorithms to check plane-stress/strain and categorize lbc *
clbc3.c	* algorithms to classify the lbcs ext/rev, mirror planes *
inv.c	* matrix inversion and vector multiply routines *
na_csym.c	* circular symmetry algorithms *
na_erev.c	* algorithms for extrusion and revolution determinations *
na_symm.c	* control for automatically reorienting the solid's princ axes *
sset.c	* menu control for reviewing/altering previous adept settings *

TABLE 4. CONTENTS OF DIRECTORY MISC

applbc.c	* Control transfer of lbcs to mesh from features *
applbc2.c	* Classify nodes of the solid *
applbc3.c	* Distribute forces and pressures on features of the solid *
cosol.c	* Reduce solid if desired. Generate new octree box *
divfrc.c	* Determine if the lbcs are mirrored and scale them *
drw_new.c	* Draw the force, pressure, and boundary condition vectors *
drw_stuf.c	* Draw basic primitives used by drw_new routines *
elm2fea.c	* Associate elements of the mesh with simple features *
elm_div.c	* Check compatibility of mesh rankings and divide the elements *
fea_dens.c	* Form lbc sphere and assign feature density *
menu.c	* Main parameters menu checks resources and purpose *
mk_mesh.c	* On exit of clips generate the mesh, transfer loads and bcs *
purpose.c	* Determine purpose of analysis and assert associated facts *
region.c	* Assign ranking to simple and complex features *
resorces.c	* Generate menu for obtaining resource information for analysis .*
s_mparm.c	* Review (elem type, surf/solid, etc.) prior to mesh generation *
s_mparm2.c	* Calculations for placing octree and choosing mesh density level *